SERGE BOISVERT - ANDRÉ DUGAS - DENISE BÉLANGER

OBLING: A TESTER FOR TRANSFORMATIONAL GRAMMARS

1. INTRODUCTION

Transformational grammars have developed with recent research in linguistics. They appear to be a powerful and explicit device for characterizing the description of sentences; they also meet conditions of adequacy that can be applied to check that a sentence, or a set of rules, is well-formed.

A transformational grammar tester is part of a strategy for the selection of a well-formed grammar matching the data base. To put it more explicitly, a tester of this sort should provide the linguist a class of the possible grammars which concerns precisely the linguistic theory. These grammars have the form given by CHOMSKY in *Aspects* (see bibliography).

2. GENERAL DESCRIPTION OF THE SYSTEM

OBLING is a program for dealing with the structures of the French language: it performs the verification of phrase structure rules, the derivation of sentences according to the transformational component and the graphic illustration of the intermediate or final structures.

In the program, LING is the routine that controls all the subroutines and the matching of the input structures with those allowed by the phrase structure rules. If the matching is impossible, a comment is

Acknowledgments. This work was supported in part by Canada Council of Arts grants #69-0404 and #71-0819.

We are also indebted to the staff of the Computer Center of the Université du Québec à Montréal for providing computing facilities, and giving this project high priority.

David Sankoff, of the Université de Montréal, is also responsible for the first version of the tree editing program. Finally, Josselyne Gérard helped debugging linguistic rules.

made. Otherwise, the output gives the graphic illustration of the tree for this sentence, or the input structures are immediately processed using transformational rules. For example,

General transformational rules are operated by a number of subroutines of which the main are explained hereafter.

3. GENERAL CAPACITIES OF THE SYSTEM

The system OBLING is divided into four parts: a main program LING, the service subroutines, the phrase structure grammar tester and the transformational grammar testers. LING and the service subroutines are stored in the central memory while the two grammars testers operate on disks.

The main program invokes the various linguistic rules and controls the application of these rules to the data base structure(s) or the derived structure(s). The service subroutines are called by the routines concerning the application of the transformational rules and work in parallel with LING during the processing.

Fig. 2. The OBLING system

4. SPECIFIC CAPABILITIES OF THE PROGRAM LING

The program LING will first initialize the working areas. Then, it loads and operates the program VERIFICATEUR which, after the reading and the verification of the input data, returns control to LING.

LING will then load and execute, using an overlay technique, the small control programs CYCLI Q1, CYCLI Q2, ..., CYCLI Qi. Each of these handles, in conjunction with LING, the mapping on the input structure of a fixed number of transformation rules. In the current version of the program, CYCLI Q1 deals with the linguistic transformational rules T_1 to T_5 included, CYCLI Q2 the rules T_6 to T_{10} included, etc. The total number of these control programs CYCLI Q depends on the memory space allowed; processing is most efficient if the number of these control programs is as small as possible.

5. INFORMATION PATTERN BETWEEN LING AND VERIFICATEUR

When VERIFICATEUR (the phrase structure grammar tester) is in memory, the structure to be analysed is read from the standard input unit (punched cards) and is processed by the subroutine CKARBRE to

Fig. 3. The VERIFICATEUR program (see figure 1 for updated tree and structure)

be validated. This subroutine first checks if the phrase structure is consistent, then calls up `CKC` which tests the names of the constituents describing the structure; finally, it compares this structure with those allowed by the phrase structure rules.

When errors are discovered during the processing, various sorts of comments are printed and followed if possible by a partial or full tree of the sentence. When updating is done, the tree is printed and the program `VERIFICATEUR` passes control to `LING`. The following illustrations concern first, the program `VERIFICATEUR` and second, an example of an updated tree and structure.

6. INFORMATION PATTERN BETWEEN LING AND THE TRANSFORMATIONAL GRAMMAR TESTERS

Each time `LING` receives the control from `VERIFICATEUR`, that is, when no further errors have been detected, it loops in order to call successively the monitors `CYCLI Q1`, ..., `CYCLI Q9` which contain up to 45 different rules; we suppose that we are working now with a specific version of a grammar.

The first of these monitors has the following structure.

<i>Transformational rule # 1</i>
<i>Transformational rule # 2</i>
<i>Transformational rule # 3</i>
<i>Transformational rule # 4</i>
<i>Transformational rule # 5</i>

Fig. 4. The `CYCLI Q1` program

When `CYCLI Q1` gets control, it is bound to the application of T_1 , ..., T_5 which correspond to the first five transformational rules; then control is switched to `LING` which calls `CYCLI Q2`. The programs `CYCLI Qn` process cyclic rules and the output structure is the input structure for the following rule. When all the cyclic rules have been applied to the input structure, `LING` starts over again at `CYCLI Q1`. If no modifications

to the already processed structures occur, or if new errors are discovered, control returns to LING.

After all the cyclic rules have been applied, the post-cyclic rules are processed in a similar manner: CYCLI QA comprises the first five post-cyclic rules CYCLI QB, the five following, and so on.

This chart illustrates the general interaction between the programs for the processing of cyclic or post-cyclic rules.

Fig. 5. Flow of Control between control programs under the direction of LING

7. SERVICE SUBROUTINES

They are implemented within the main monitor LING. All but a few of these subroutines are called during the execution of the routines corresponding to the 88 rules, that is during the phrase structure analysis or the mapping of n structures.

A short description of the main subroutines follows: ARBRE (tree). This subroutine is responsible for printing the tree. At the input, we find a vector D of ND elements which represents the tree. The horizontal distance for printing is calculated along with the total number and the relative position of these nodes; the vertical one is fixed.

For example,

TACHE* 2 *

FIGURE 6a

NOMBRE DE NOEUDS 21

NOMBRE DE NOEUDS TERMINAUX 12

CHAÎNE

```

1 = $ , D(1) = 20
2 = LE , D(2) = 13
3 = N , D(3) = 15
4 = V , D(4) = 16
5 = $ , D(5) = 20
6 = $ , D(6) = 21
7 = PRP , D(7) = 17
8 = QUE , D(8) = 17
9 = LE , D(9) = 14
10 = N , D(10) = 18
11 = V , D(11) = 19
12 = $ , D(12) = 21
13 = DET , D(13) = 15
14 = DET , D(14) = 18
15 = GN , D(15) = 20
16 = GV , D(16) = 20
17 = C , D(17) = 21
18 = GN , D(18) = 21
19 = GV , D(19) = 21
20 = P , D(20) = -1
21 = P , D(21) = -1

```

ARBORESCENCE NON PRODUITE SUR DEMANDE : AUCUNE

RÈGLES IGNOREES SUR DEMANDE : AUCUNES

Fig. 6a. Representation of the tree in memory

7.1. OTE (*Remove*).

This subroutine is needed when nodes are erased; another subroutine, NEWTREE will erase the nodes. In the example below, OTE sets $D(6)$, $D(7)$, $D(13)$ to zero, and NEWTREE erases nodes numbered 6, 7 and 13. If node 12 was also erased, OTE and NEWTREE would have erased node 28 automatically. The same holds for the node 32, where all the nodes between 6 and 13 would have been erased.

7.2. DFER, DFERX, GFER, GFERX.

Except for a few details, these four subroutines do the same work. For example, DFERX $[I, J]$ is applied to a structure J that has to be moved to the right under a dominating node I . As illustrated below, DFERX $[31, 30]$ moves the structure headed by 30 to the right under the node

Fig. 7b. Sample tree where OTE and NEWTREE have applied

31. (Node 31 was created by rule *T2* and *DFER* was applied on the resulting tree)

<i>DFER</i>	$[I, J]$	makes node <i>J</i> the next younger brother of node <i>I</i>
<i>DFERX</i>	$[I, J]$	makes node <i>J</i> the youngest son of node <i>I</i>
<i>GFER</i>	$[I, J]$	makes node <i>J</i> the next older brother of node <i>I</i>
<i>GFERX</i>	$[I, J]$	makes node <i>J</i> the oldest son in node <i>I</i>

The general technique for these four subroutines is the following. Before modification, the tree is copied in another area of memory. All the terminal nodes identified with the structure *J* take the place of the terminal nodes identified with the structure *I*. Then, the terminal nodes of *I* are requested in the copied structure and parsed with their dominating nonterminal nodes at the right place. *GFER* permits the new numbering of the sequence and, if necessary, prunes the tree.

In the example illustrated below (Fig. 9*a* and 9*b*), *GFER* [14, 13] is applied and node *CPL* (13) has been attached to node 16, the father of node 14. If *GFERX* [14, 13] had been specified, node *CPL* would have been attached directly to node 14, rather than the father of node 14.

7.3. *INTERV.*

This subroutine is used for the permutation of 2 structures. For example, *INTERV* [*I, J*] where *I* = 24 and *J* = 28 gives rise to the structural change illustrated below.

7.4. *INSERT.*

This subroutine is used for the insertion of a new terminal node; for example, *INSERT* [4, 1*HE*, 1*HT*, 1*HR*, 9] introduces node with name *ETR* which becomes a new son of node 9.

7.5. *Other subroutines.*

There is a number of other subroutines concerning conditions specified within a rule, such as the presence or absence of a node or of a feature in the whole structure, the logical restrictions of loose equality or strict equality.

*

Fig. 10a. *Input structure*

Fig. 11b. *Insertion of a node (after)*

8. CONCLUSIONS

OBLING is a system which has been implemented in low-level FORTRAN IV for the CDC 6400. It occupies 55,000, 60-bit words of memory. It has about 7000 lines of comments and instructions.

REFERENCES

- N. CHOMSKY, *Aspects of the Theory of Syntax*, Cambridge (Mass.), 1965.
- A. DUGAS, et al., *Description syntaxique élémentaire du français inspiré des théories transformationnelles*, Montréal, 1969.
- J. FRIEDMAN, *A Computer Model of Transformational Grammar*, New York, 1971.
- D. LIEBERMAN, (ed), *Specification and Utilization of a Transformational Grammar*, Yorktown Heights (N.Y.), 1966.
- D. L. LONDE, W. J. SCHOENE, TGT: *A Transformational Grammar Tester*, in *Proc. Spring Joint Computer Conference*, Part I, 1968, pp. 385-393.
- R. PETRICK, *A Recognition Procedure for Transformational Grammars*, Ph.-D. Dissertation, Cambridge (Mass.), 1965.
- J. R. ROSS, *A proposed rule of tree-pruning*, NSF-17, Computation Laboratory, Harvard University, IV (1966), pp. 1-18.
- A. M. ZWICKY, J. FRIEDMAN, B. HALL, D. E. WALKER, *The MITRE Syntactic Analysis Procedure for Transformational Grammars*, in *Proc. Fall Joint Computer Conference*, Vol. 27, Pt. 1, 1965, pp. 317-326.